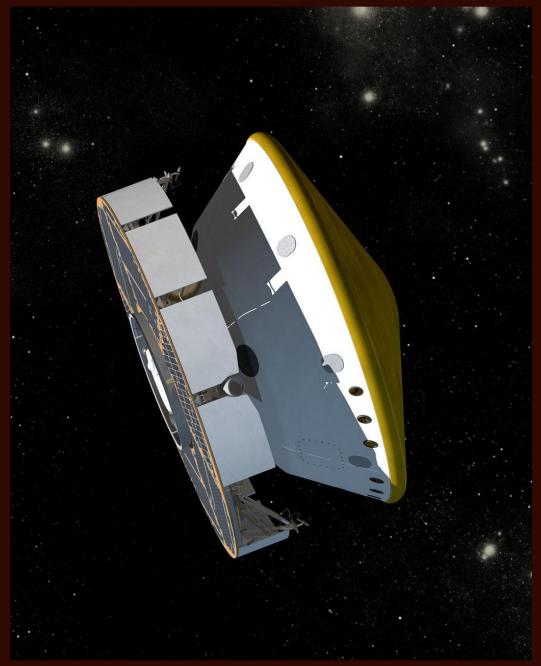
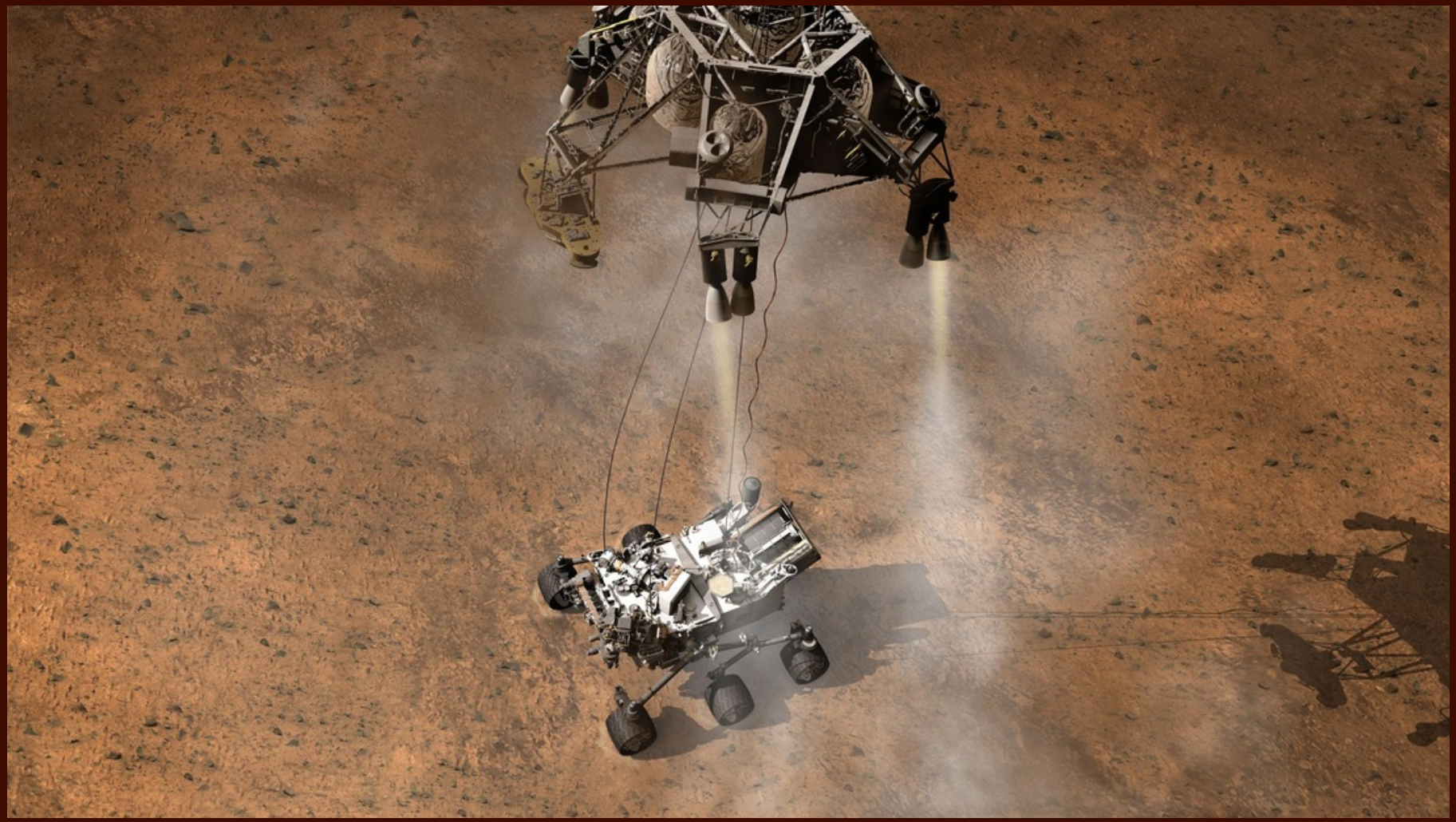
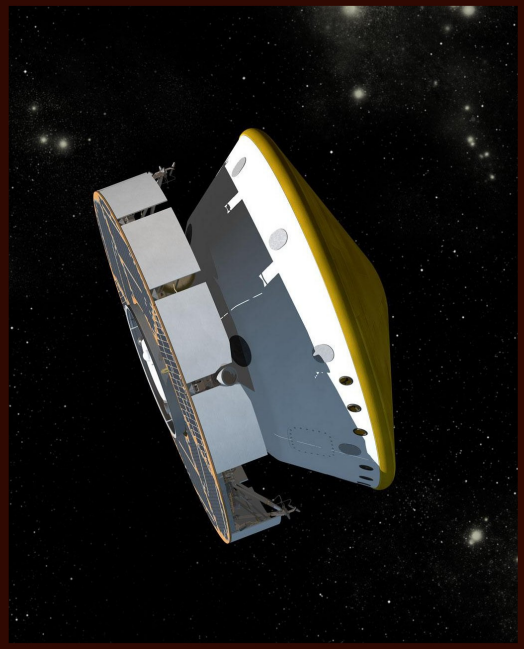


Managing data for Curiosity, fun and profit

Rajeev Joshi
NASA / Jet Propulsion Laboratory



3 missions in one



Cruise

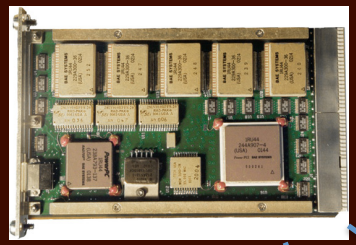
EDL

Entry, Descent & Landing

Surface

Avionics

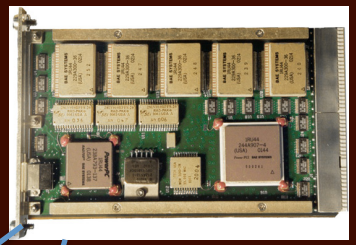
Prime Computer



RAM

Flash memory

Backup Computer



Flash memory

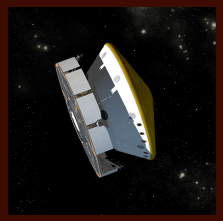
RAM

X-Band radios

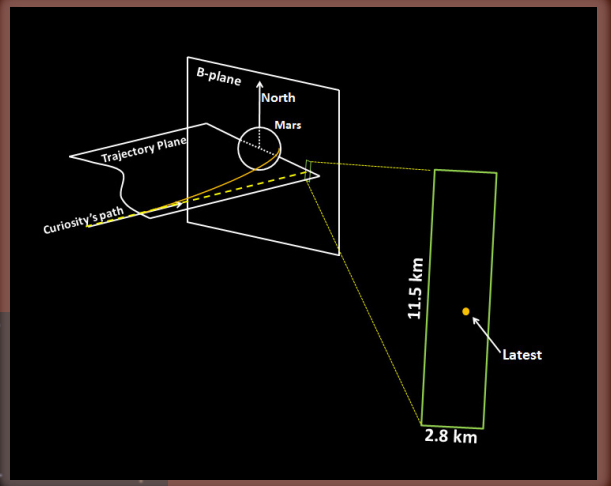
UHF radios

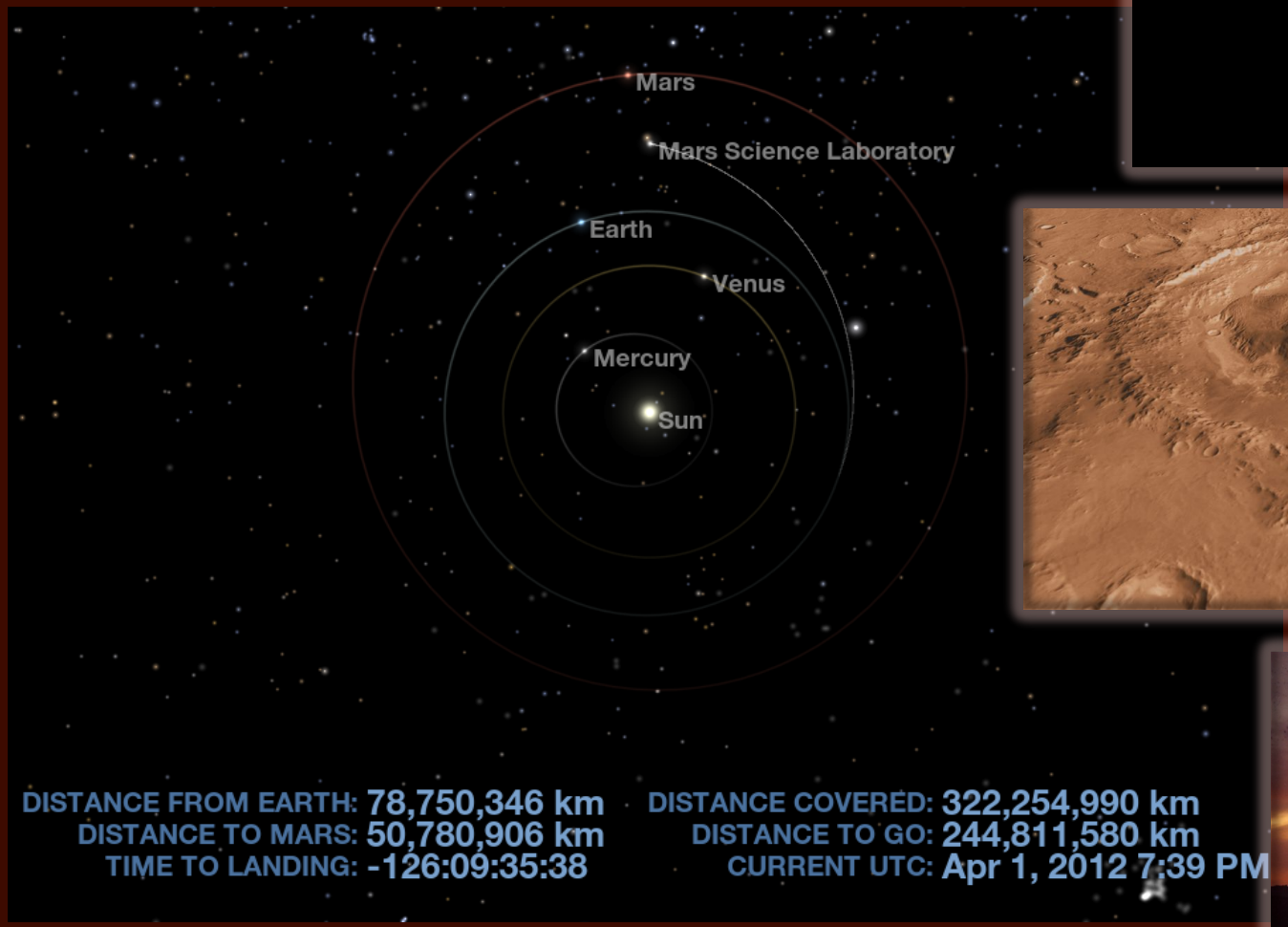
direct-to-Earth
low bandwidth
100s of bps

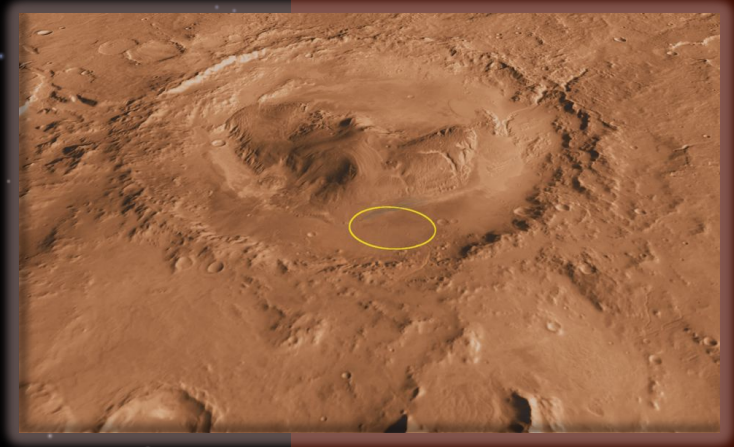
rover-to-orbiter
high bandwidth
2 Mbps



Cruise
8.3 months



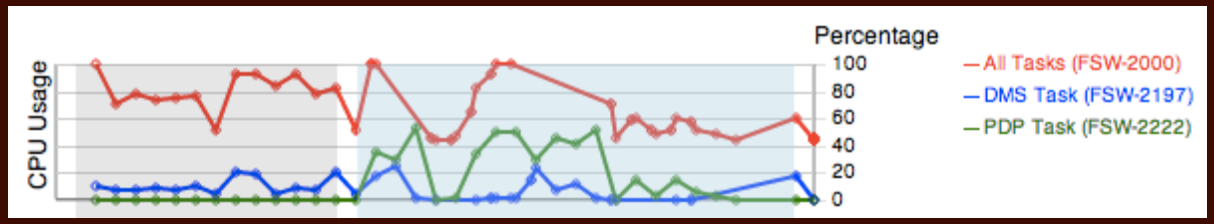




“Realtime” telemetry

Streamed continuously whether or not anyone is listening

Periodic measurements (“EHA”)



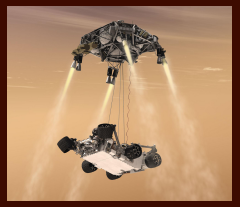
Discrete event log (“EVRs”)

ERT	SCLK	SOURCE	TYPE	DATA
2012-206T00:35:07.668	0396361199.90137	FSW RT	EVR DIAGNOSTIC	Delivering 3 bytes of VC-1 uplink data targeted for String A on S/C ID 76.
2012-206T01:18:43.038	0396361199.90137	FSW REC	EVR DIAGNOSTIC	Delivering 3 bytes of VC-1 uplink data targeted for String A on S/C ID 76.
2012-206T00:35:07.668	0396361199.90355	FSW RT	EVR COMMAND	Dispatching immediate command SEQ_LIST: stcode=0x8B1C, immediate command number=33, seco
2012-206T05:11:10.823	0396361199.90355	FSW REC	EVR COMMAND	Dispatching immediate command SEQ_LIST: stcode=0x8B1C, immediate command number=33, seco
2012-206T00:35:07.668	0396361199.90437	FSW RT	EVR DIAGNOSTIC	Successfully dispatched immediate command SEQ_LIST: stcode=0x8B1C, immediate command numb
2012-206T01:18:43.038	0396361199.90437	FSW REC	EVR DIAGNOSTIC	Successfully dispatched immediate command SEQ_LIST: stcode=0x8B1C, immediate command numb
2012-206T00:35:12.146	0396361208.13908	FSW RT	EVR WARNING_LO	Task [context Id = TASK_SEQ_CTRL_ENTRY] did not report in by the squawk threshold [6] seconds; th
2012-206T05:11:06.345	0396361208.13908	FSW REC	EVR WARNING_LO	Task [context Id = TASK_SEQ_CTRL_ENTRY] did not report in by the squawk threshold [6] seconds; th
2012-206T00:35:12.146	0396361208.52693	FSW RT	EVR COMMAND	Successfully completed immediate command SEQ_LIST: immediate command number=33.
2012-206T05:11:10.823	0396361208.52693	FSW REC	EVR COMMAND	Successfully completed immediate command SEQ_LIST: immediate command number=33.
2012-206T00:38:38.152	0396361412.84778	FSW RT	EVR ACTIVITY_HI	Current-Attitude-Error-Correction-Vector in J2000 (rad) = [1.30333e-04 -5.55175e-07 -6.46999e-06]
2012-206T00:38:38.152	0396361412.84778	FSW REC	EVR ACTIVITY_HI	Current-Attitude-Error-Correction-Vector in J2000 (rad) = [1.30333e-04 -5.55175e-07 -6.46999e-06]
2012-206T00:38:38.152	0396361412.84785	FSW RT	EVR ACTIVITY_HI	Attitude quaternion q_Aka_J = [3.45925e-01 -6.44945e-01 -2.01949e-01 6.50844e-01]
2012-206T00:38:38.152	0396361412.84785	FSW REC	EVR ACTIVITY_HI	Attitude quaternion q_Aka_J = [3.45925e-01 -6.44945e-01 -2.01949e-01 6.50844e-01]

“Recorded” telemetry and data products

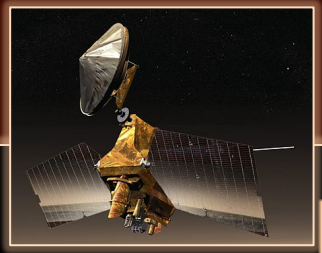
Realtime telemetry saved to nonvolatile memory
Sent only when requested by ground

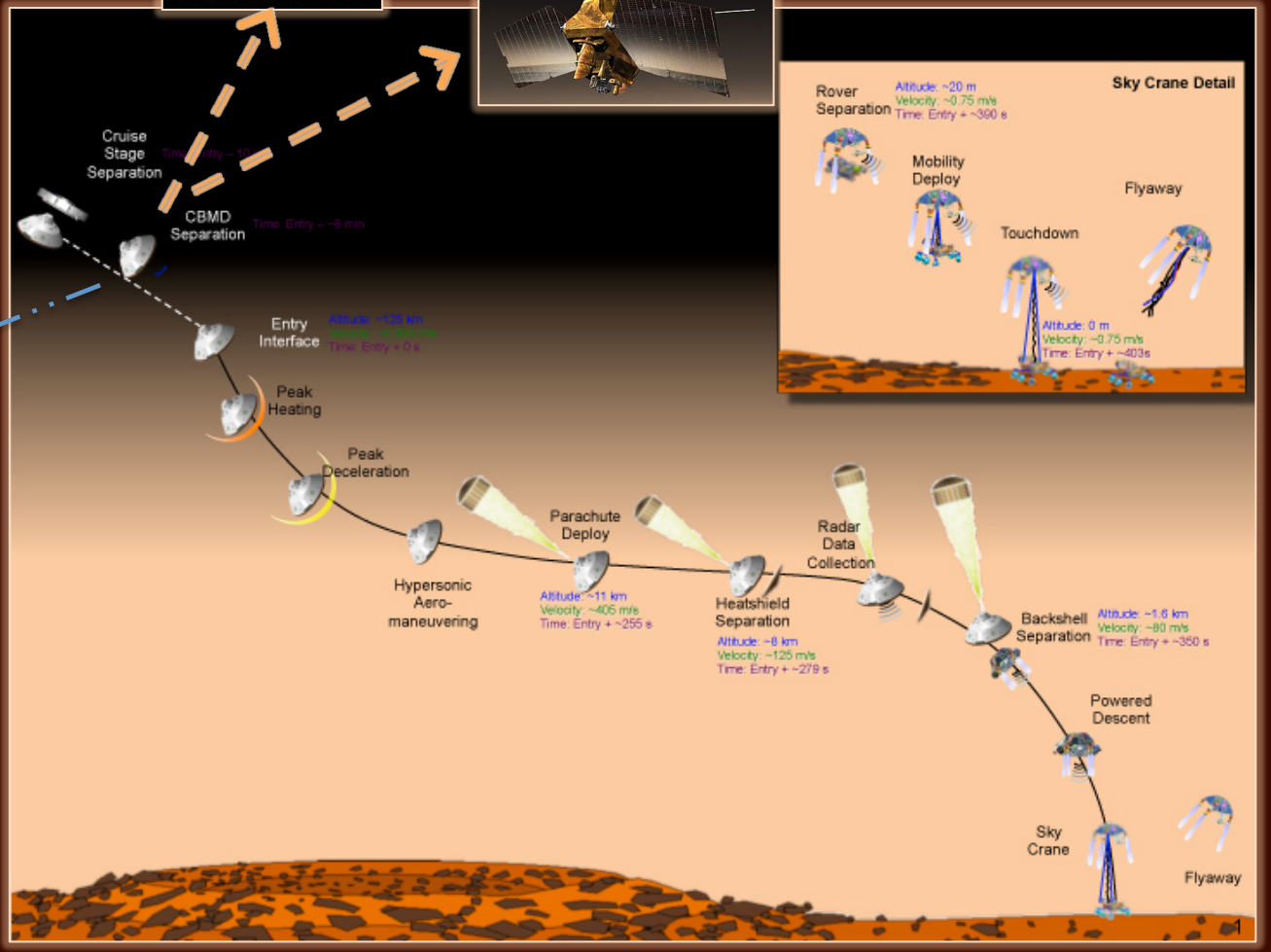
“Data products” containing
detailed engineering data
science observations (images, spectra, …)



EDL

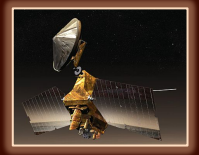
7 minutes

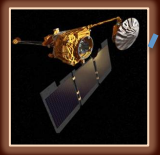


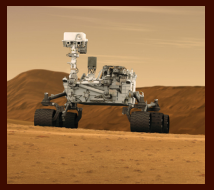


Surface

>2 years







10:00 16:00 02:00 10:00

Typical sol on surface

-- wake up and listen to Earth --
10:00 – 10:40 X-Band window with Earth; receive plan
-- execute plan --
16:00 – 16:10 UHF PM window with MRO
17:05 – 17:19 UHF PM window with Odyssey
-- rover asleep –
02:00 – 02:10 UHF AM window with MRO
-- rover asleep --
03:22 – 03:33 UHF AM window with Odyssey
-- rover asleep –
-- wake up and listen to Earth --

Data Management Needs

Cruise

store data to flash on generation
retrieve data from flash on ground command

EDL

stream data to orbiters as it is being generated
commit data to flash before surface transition

Surface

store data to flash on generation
retrieve data from flash in time for comm windows

Challenges

Processor speed 132 Mhz

Main memory 128 MByte (same as MER)

Implement a reliable storage mechanism
on unreliable (flash) medium

Predictable behavior
bounds on worst-case performance (time and memory)
even in the event of an unexpected reboot

Examples *wakeup and prep for AM comm windows*
support upto 250,000 products in system

Challenges

Processor speed 132 Mhz

Main memory 128 MByte (same as MER)

Implement a reliable storage mechanism
on unreliable (flash) medium

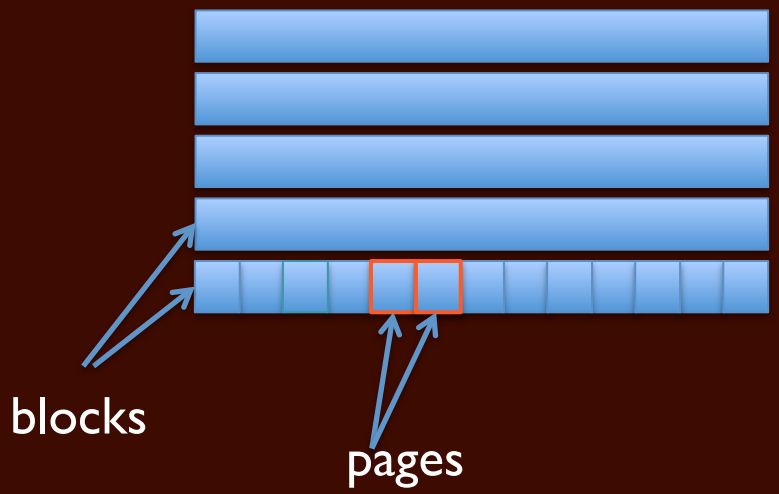
Predictable behavior
bounds on worst-case performance (time and memory)
even in the event of an unexpected reboot

Examples *wakeup and prep for AM comm windows*
overhead with 250,000 products in system

Why flash filesystems are hard

Asymmetry of write / erase

write_page
erase_block



Bad blocks

Limited block lifetimes

must do *wear-leveling*

Challenges

Processor speed 132 Mhz

Main memory 128 MByte (same as MER)

Implement a reliable storage mechanism
on unreliable (flash) medium

Predictable behavior
bounds on worst-case performance (time and memory)
even in the event of an unexpected reboot

Challenges

Processor speed 132 Mhz

Main memory 128 MByte (same as MER)

Implement a reliable storage mechanism
on unreliable (flash) medium

Predictable behavior
bounds on worst-case performance (time and memory)
even in the event of an unexpected reboot

Hard deadlines!

Formal Methods?

Filesystem specification

```
creat()
open()
read(),write()
lseek()
mkdir()
rmdir()
```

use pre-/post- conditions

"Reset-reliability"

filesystem shall remain consistent across an unexpected reboot

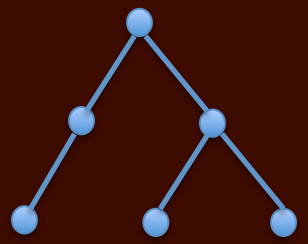
strengthen the spec to require

operations be atomic wrt reboot

Difficulties

A filesystem is a recursive data type

requires reasoning about reachability (hard)



Initial attempts

TLA+ (Joshi)
ACL2 (Erickson)
VDM (Hu)

But

large semantic gap between spec and flight code
lack of available tools to bridge this gap

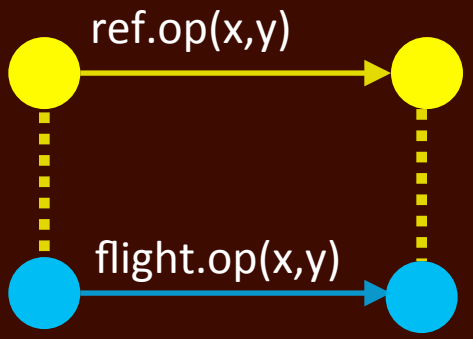
Filesystem reference implementation (Holzmann)

1000 lines of C vs 6000 lines of flight code

Simpler

- assumes simple storage medium (volatile RAM)
- not fault tolerant

Filesystem correctness

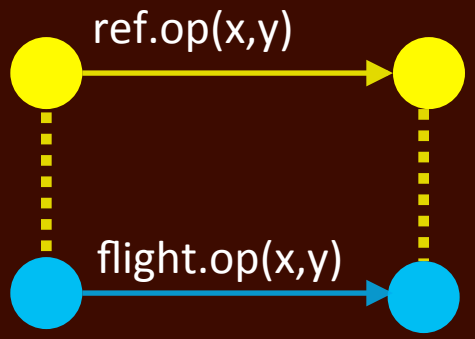



```
      choose op(x,y)
       [ inject fault in flight ]
      r0 := flight.op(x, y)
      r1 := ref.op(x, y)
(a)   assert   r0 == r1
(b)   assert   flight.state   ~   ref.state
(c)   assert   flight.invariants()
```

tree equivalence

includes space- and wear-leveling checks

Filesystem correctness




```
      choose op(x,y)
       [ inject fault in flight ]
      r0 := flight.op(x, y)
      r1 := ref.op(x, y)
(a)   assert   r0 == r1
(b)   assert   flight.state   ~   ref.state
(c)   assert   flight.invariants()
```

tree equivalence

data structure invariants; properties not modeled in ref

Golden Rule for testing

if either (a) or (b) fails

strengthen (c) until it also fails

Randomized testing

Simple to set up and maintain

- we used SPIN as a test driver (with unsound abstractions)
- instrument with CIL to measure *path coverage* to evaluate heuristics

Easy to parallelize

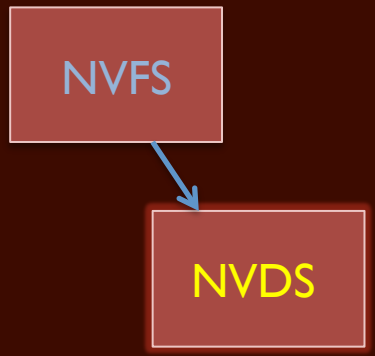
Surprisingly effective

- found many bugs that would not have been caught in system test

The filesystem caching bug

- adding read cache to NVDS
- exposed a latent bug in NVFS

would have resulted in all file creations failing after ~80 sols on surface



Failures of the test program

Sol-217 anomaly

a bug in filesystem rebuild on boot
caused a file to become corrupt (wrong size)
→ failed checksum
resulted in safe mode (loss of 1 day)

Scenario
latest file F is deleted
-- reboot --
new file G is created and
reuses pages from F
-- reboot --

Sol-200 anomaly

catastrophic failure of flash memory resulted in filesystem unavailability
manager task did not handle situation gracefully
task that controls shutdown became blocked
required ground to force swap to backup computer

Randomized testing challenges

Overnight run returns many occurrences of the same bug

What to do when no more bugs are being found?

Less effective without a reference implementation

Randomized testing challenges

Overnight run returns many occurrences of the same bug

What to do when no more bugs are being found?

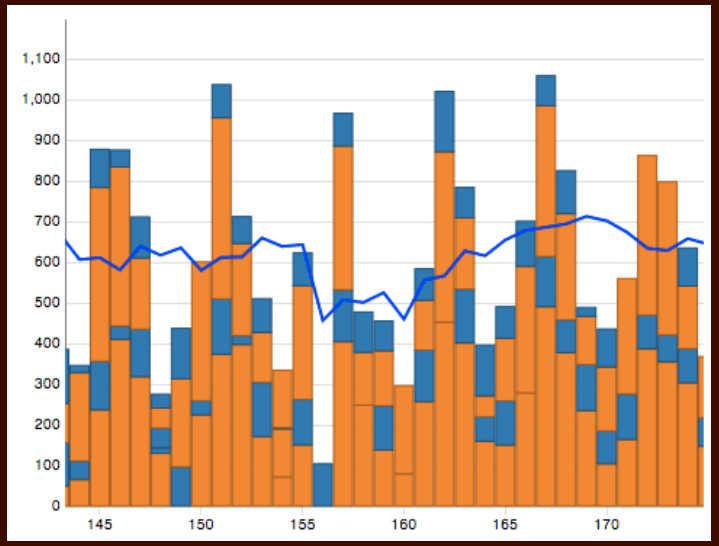
Less effective without a reference implementation

How do we climb the verification ladder
from randomized testing
to full functional verification?

Formal Methods & Ground Operations

> 150 Gbits of data returned by MSL so far

often >1 Gbit in a single day



how do we analyze all this data?

- anomaly investigations
- trending reports
- find smoking guns

Telemetry Analysis

current practice

- ad-hoc scripts (python, perl, Excel macros)

hard to understand, maintain, debug

Telemetry Analysis

current practice

ad-hoc scripts (python, perl, Excel macros)

hard to understand, maintain, debug

Can we leverage formal methods?

declarative rules

process telemetry 24 / 7

semantic querying to aid anomaly investigations